Stochastic Analysis of BPMN with Time in Rewriting Logic

Francisco Durán, Camilo Rocha, Gwen Salaün

Stochastic Analysis of BPMN with Time in Rewriting Logic

Francisco Durán

University of Málaga, Málaga, Spain

Camilo Rocha

Pontificia Universidad Javeriana, Cali, Colombia

Gwen Salaün

Univ. Grenoble Alpes, CNRS, Grenoble INP, Inria, LIG, F-38000 Grenoble France

Abstract

A business process is a set of structured activities that provide a certain service or product. Business processes can be modeled using the BPMN standard, and several industrial platforms have been developed for supporting their design, modeling, and simulation. This paper presents a rewriting logic executable specification of BPMN with time and extended with probabilities. Duration times and delays for tasks and flows can be specified as stochastic expressions, while probabilities are associated to various forms of branching behavior in gateways. These quantities enable discrete-event simulation and automatic stochastic verification of properties such as expected processing time, expected synchronization time at merge gateways, and domain-specific quantitative assertions. The mechanization of the stochastic analysis tasks is done with Maude's statistical model checker PVeStA. The approach is illustrated with a running example and further experimental results encompass specifications from the literature.

Key words: Business processes, BPMN, timing behavior, probabilities, rewriting logic, executable specification, automated verification, stochastic analysis, statistical model checking, Maude.

1. Introduction

In the last ten years, business process management has become a strategic activity in organizations because streamlining the use of time and resources has the potential to increase profit margins. A business process consists of a set of structured tasks that together provide a certain service or product by guiding the accomplishment through time of activities such as invoicing management, production lines, after-sale services, and wage payments. BPMN, the *Business Process Model and Notation* [28], is a workflow-based graphical language that has emerged as the *de-facto* standard for business process definitions. State-of-the-art tools for BPMN focus on the description, implementation, and simulation of business processes. However, these tools provide very little support for understanding the quantitative aspects in process design.

When modeling a BPMN process many crucial questions arise from a correctness and optimization point of view. For example: is the workflow precisely modelling what it is intended? Is the workflow free of errors and bugs? Are certain properties of interest preserved? What is the degree of parallelism of the process? Are there bottlenecks and, if so, where? What is the average execution time of the workflow? How long will the workflow wait at some specific synchronization points? Formal verification and optimization of business processes aim at, respectively, ensuring the correct behavior and improving these processes, by effectively answering the above questions, with the final goal to reduce costs and augment efficiency. Nonetheless, process correctness and optimization are far from being simple questions to answer, particularly when modeling complex combinations of tasks, nesting of gateways, cyclic behavior, and quantitative aspects such as probabilities.

This paper presents modeling and automated analysis techniques for the quantitative evaluation of timing behavior in BPMN. Towards this goal, the BPMN language – which provides a number of different building blocks – is enriched with stochastic expressions for specifying time (duration and delay) and probabilistic branching. A process specification makes use of probability distribution functions for defining the execution time of tasks or flows, and of probabilities for defining the likelihood of branching executions at split gateways. The idea is that a process designer can use these constructs when defining a process and then use the type of automatic verification presented in this paper to quantitatively estimate its correctness and performance. The output of the verification task is helpful for validating and optimizing a process design (e.g., structure, precision, message buffer dimension, and use of resources), before it is deployed in a pro-

duction environment.

The approach relies on an executable specification of BPMN in rewriting logic [37], a logic of concurrency that naturally deals with states and concurrent transitions between them. This specification is formulated as a probabilistic rewrite theory [3], parametric in a family of probability distribution functions, that supports a broad class of BPMN constructs and stochastic expressions. When executed, a probabilistic rewrite theory simulates discrete probabilistic choice as found in discrete-time Markov chains and stochastic continuous-time as found in continuous-time Markov chains [46]. The executable specification of BPMN in rewriting logic supports: (i) tasks and flows whose times are constant values or sampled from several probabilistic distribution functions, (ii) probabilistic inclusive and exclusive gateways as well as parallel gateways, (iii) loops, and (iv) unbalanced workflow structures.

The proposed automatic analysis of quantitative properties relies on the PVeStA statistical model checker [5]. Given a probabilistic rewrite theory, such as the one presented in this paper, PVeStA is used to simulate its execution while automatically evaluating expected values of any numerical expression or path expression encoded in the QuaTEx language [3]. This is achieved automatically by performing enough Monte Carlo simulations to meet an error threshold and still be meaningful for statistical inference. This approach is used to compute several properties of interest on a process such as the expected processing time (including variance, minimum, and maximum processing times collected during the sampled runs) and the synchronization times (i.e., waiting times) at merge gateways. Such capabilities are illustrated in this paper with the help of a running example. The output of the quantitative analysis is used as part of a discussion illustrating how the running example can be improved by reducing expected times. Additional results on the statistical model checking of real-world business processes from the literature are also included.

To sum up, the main contributions of this paper are the following:

- an executable specification in rewriting logic of a subset of BPMN (including inclusive splits, loops, and unbalanced gateways) extended with support for stochastic specification of time and branching constructs;
- automated verification of stochastic properties of interest such as expected processing time of a process and synchronization time at merge gateways using statistical model checking; and
- validation of the approach by application to several real-world BPMN pro-

cesses.

The organization of the paper is as follows. Section 2 introduces BPMN extended with timing and branching probabilistic information. Section 3 presents some preliminaries on probabilistic modeling and stochastic analysis in rewriting logic. Section 4 gives a summary of the executable specification of the subset of BPMN considered here. Section 5 presents how performance evaluation of BPMN processes is achieved using the PVeStA tool and experiments on several real-world examples. Section 6 surveys related work and Section 7 concludes the paper.

Additional details on the specifications and examples presented throughout the paper are available to the reader at the accompanying webpage [18].

2. BPMN with Time and Probabilities

This section introduces the subset of BPMN considered in this paper, which focuses on behavioral aspects (start/end events, tasks, flows, gateways) enriched with time and probabilities. A process modeling a business trip organization in BPMN with timing and probability annotations is introduced with the purpose of illustrating these concepts, and serving as a running example in the upcoming sections.

2.1. BPMN Overview

A BPMN process is a directed graph with nodes as vertices and sequence flows as directed edges. A node is a start or end event, a task, or a gateway. Start and end events are used to initialize and terminate processes, respectively. A task represents an atomic activity, and has exactly one incoming and one outgoing flow. A gateway is used to control the split patterns (i.e., flow divergence) and merge patterns (i.e., flow convergence) of execution in a process. In this paper, a process is considered to have exactly one start event and at least one end event. The three main gateways available in BPMN are considered, namely, exclusive, parallel, and inclusive gateways. An exclusive gateway chooses one out of a set of mutually exclusive alternative incoming or outgoing branches. A parallel gateway creates concurrent flows for all its outgoing branches or synchronizes concurrent flows for all its incoming branches. In an inclusive gateway, any number of branches among all its incoming or outgoing branches may be taken. For any kind of gateway considered here (exclusive, parallel, inclusive), there is a split and a corresponding merge pattern. Figure 1 summarizes the syntax of BPMN supported in this work,

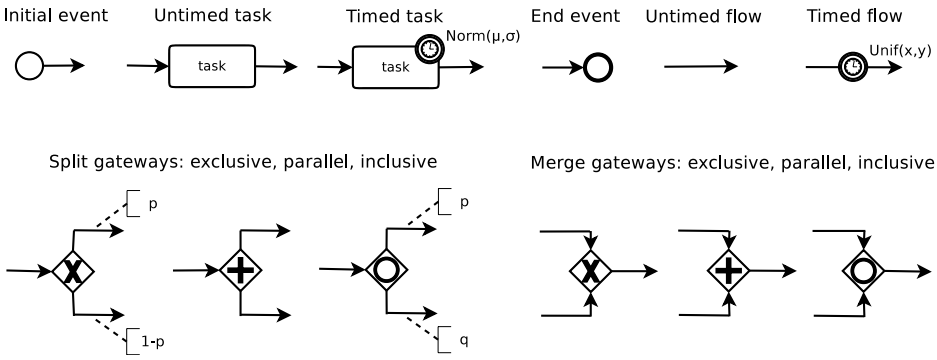

Figure 1: BPMN syntax augmented with timing behavior and probabilities

including examples of the timing and probability constructs that are explained later in this section.

The semantics of BPMN is described in official documents [28, 39] and some attempts have been made to formalize its semantics (e.g., [17, 36, 42, 49]). The execution semantics of BPMN is usually given by means of tokens representing how the execution of the process evolves over time. At the beginning of the process execution, there is exactly one token at the start event. A token can move along a number of sequence flows. A token can also enter and leave a task by following the flow associated to that task. When a token arrives at a gateway, the execution behaves differently depending on the kind of gateway encountered. When a token arrives at a parallel split gateway, the token is consumed and one token is generated for every outgoing flow of the split gateway. When a token is consumed at an exclusive split gateway, only one token is created and assigned to one of its outgoing flows. In the case of an inclusive split gateway, when a token is consumed, some new tokens are generated and assigned to the outgoing flows (at least one, at most all). For exclusive and inclusive split gateways, the choice of outgoing branches to be activated depends on data-based conditions (e.g., "$x > 50$" is associated to one outgoing flow and "$x \leq 50$" is associated to the other flow) that can be evaluated to true or false. In our work, we model such conditions as probabilistic choices. As for merge gateways, they usually act as synchronization points and can be triggered when all expected tokens have arrived at this merge gateway. A process finishes execution when all tokens have reached an end event.

2.2. Timing Behavior and Probabilities

The class of BPMN processes considered in this paper is parametric in a family of probability distribution functions specifying the time of tasks and flows (duration). Stochastic expressions are assigned to a BPMN task or flow for specifying its duration in time units. In the presence of timed tasks and flows, the execution semantics of a BPMN process is as follows: a token has a run-to-completion time at its current position and it needs to wait until this time becomes zero before advancing in the flow. Probabilities offer a useful way of describing the behavior of split gateways without explicitly associating data-based conditions to flows. Probabilities can be associated to outgoing flows of exclusive and inclusive split gateways, denoting how an associated flow is likely to be assigned a newly created token during execution.

Specifically, a time value can either be a constant value (a non-negative real number, possibly 0) or sampled from a probability distribution function according to some parameters. The probability distribution functions available for the executable specification of BPMN include the exponential, Weibull, normal/Gauss, Γ, χ^2, Erlang, F, geometric, Pascal, Pareto, binomial, and uniform functions [48]. In Figure 1, normal and uniform distribution functions are associated to some tasks and flows, respectively. The expression $Norm(\mu, \sigma)$ means that the time value for the corresponding task or flow is to be sampled from the normal distribution function with mean μ and variance σ. Analogously, the expression $Unif(x, y)$ means that the time value for the corresponding task or flow is to be sampled uniformly from the interval $[x, y]$. In the executable specification of BPMN presented in this paper, there is a signature specifying the parameters required to sample values for each one of the probability distribution functions listed above.

The probabilities associated to the outgoing flows of an exclusive split must sum up to 1. However, each outgoing flow in an inclusive split can be equipped with a probability between 0 and 1 without a restriction on their total sum, because several branches can be triggered in the case of an inclusive split gateway (every branch is independent of each other).

The timing behavior and the split probabilities introduced in this section are treated as annotations in the BPMN process. On a wider scale, it is assumed that BPMN processes are syntactically correct. This can be ensured by using tools such as the Activiti BPM platform, Bonita BPM, or the Eclipse BPMN Designer. BPMN processes are then automatically transformed to a Maude specification using a a plugin implemented as an extension of the VBPMN platform [31] (see Section 4).

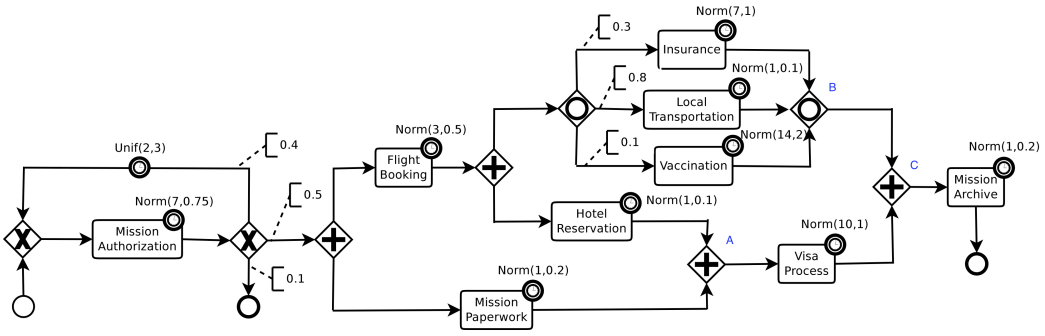

Figure 2: A business trip organization process in BPMN

2.3. Running Example

The running example, depicted in Figure 2, is a process detailing the steps required to prepare a business trip within an organization. The process starts by asking for an authorization to organize a trip. Three continuations are possible: the process abruptly terminates, the authorization is to be requested again with, e.g., additional information, or the trip is accepted and then the rest of the process is triggered. In the latter case, the process continues by reserving flight tickets and carrying out mission paperwork. Once the flight tickets are issued, accommodation reservation and other additional activities (e.g., insurance, vaccines) are performed in parallel. The visa process is initiated only when all reservations (i.e., flights and hotels) are ready and when the paperwork is completed. When all the prerequisites for the trip are satisfied, the mission details are stored in a specific database.

Quantitative information is used in several parts of the business trip organization process by associating: (i) timing behavior to tasks and flows, and (ii) probabilities for triggering outgoing flows in case of choice (i.e., exclusive and inclusive split gateways). These expressions can be defined by internal rules of the organization (e.g., authorization is delivered within a week) or by contextual information (e.g., 50% of trips are authorized) at design time. To be able to refer to the parallel and inclusive merge gateways in this process in Section 5, we have identified them in Figure 2 with upper case letters (A, B, C). Note that exclusive, inclusive and parallel gateways, looping behavior and unbalanced structure of workflows are present in this process.

3. Stochastic Analysis in Rewriting Logic: An Overview

dfe

Rewriting logic [37] is a semantic framework that unifies a wide range of models of concurrency. Specifications in rewriting logic are called rewrite theories and they can be executed in the rewriting logic implementation Maude [14]. By being executable, they benefit from a set of formal analysis tools available to Maude, such as state-space exploration, automata-based LTL model checking, and theorem proving.

A *rewrite theory* is a tuple $\mathcal{R} = (\Sigma, E \uplus B, R)$ with: (i) $(\Sigma, E \uplus B)$ an order-sorted equational theory with signature Σ, E a set of equations over the set T_Σ of Σ-terms, and B a set of structural axioms over T_Σ for which there exists a finitary matching algorithm (e.g., associativity, commutativity, and identity, or combinations of them); and (ii) R a finite set of (possibly conditional) rewrite rules over T_Σ. Intuitively, $\mathcal{R}$ specifies a concurrent system where states are elements of the set $T_{\Sigma/E\uplus B}$ of Σ-terms modulo $E \uplus B$ and with concurrent transitions axiomatized by the rewrite rules R [11]. For $t, t' \in T_\Sigma$ representing states of the concurrent system described by $\mathcal{R}$, a transition from t to t' is captured by a formula of the form $t \rightarrow_{\mathcal{R}} t'$. More precisely, the symbol $\rightarrow_{\mathcal{R}}$ denotes the binary rewrite relation induced by R over $T_{\Sigma/E\uplus B}$.

In the Maude framework, object-oriented systems can be specified by object-oriented modules in which classes and subclasses are declared. A class is declared with syntax `class` C `|` a_1`:`S_1`,` ... `,` a_n`:`S_n, where C is the name of the class, a_i are attribute identifiers, and S_i are the sorts of the corresponding attributes. Objects of a class C are then record-like structures of the form $< O$ `:` C `|` a_1`:`v_1`,` ... `,` a_n`:`$v_n >$, where O is the name of the object, and v_i are the current values of its attributes. Objects then evolve as result of the application of rewrite rules.

The executable specification of BPMN presented in Section 4 is a *probabilistic rewrite theory* [3]. In a standard rewrite theory, a rewrite rule has the form

$$l(\mathbf{x}) \rightarrow r(\mathbf{x}) \text{ if } \phi(\mathbf{x})$$

where the condition $\phi(\mathbf{x})$ is assumed to be purely equational. Such a rule specifies a pattern $l(\mathbf{x})$ that can match some fragment of the system's state t if there is a substitution θ for the variables $\mathbf{x}$ that makes $\theta(l(\mathbf{x}))$ equal modulo B to that state fragment, changing it to the term $\theta(r(\mathbf{x}))$ in a local transition if the condition $\theta(\phi(\mathbf{x}))$ evaluates to true. In a probabilistic rewrite theory, however, rewrite rules can have the more general form

$$l(\mathbf{x}) \rightarrow r(\mathbf{x}, \mathbf{y}) \text{ if } \phi(\mathbf{x}) \text{ with probability } \mathbf{y} := \pi(\mathbf{x})$$

Because the pattern $r(\mathbf{x}, \mathbf{y})$ on the right-hand side may have new variables $\mathbf{y}$, the next state specified by such a rule is not uniquely determined: it depends on the choice of an additional substitution ρ for the variables $\mathbf{y}$. In this case, the choice of ρ is made according to the family of probability functions π_θ: one for each matching substitution θ of the variables $\mathbf{x}$. Therefore, a probabilistic rewrite theory can express both deterministic and probabilistic behavior of a concurrent system. At any given point of execution of a probabilistic rewrite theory many different rules can be enabled. Once a matching substitution θ has been chosen for one of these rules, the choice of the substitution ρ is made probabilistically according to the probability distribution function π_θ.

PMaude [3] is both a language for specifying probabilistic rewrite theories and an extension of Maude supporting the execution of such theories by discrete-event simulation. PMaude can capture the dynamics of various elements of a system by stochastic real-time: computation and message-passing between entities of a system may take some positive real-valued time that can be distributed according to some continuous probability distribution function. Time associated to computation and message passing can also be zero, indicating instant transitions and synchronous communication. In general, PMaude supports discrete probabilistic choice as found in discrete-time Markov chains and stochastic continuous-time as found in continuous-time Markov chains.

A specification in PMaude without *unquantified non-determinism* is a key requirement for the type of statistical analysis for BPMN described in Section 5. Intuitively, non-existence of unquantified non-determinism means that non-deterministic choice during the simulation of a probabilistic rewrite theory $\mathcal{R}$ is exclusively due to probabilistic choice and not to concurrent transitions firing simultaneously at different parts of a system state. Under this assumption (and some admissibility assumptions on $\mathcal{R}$), a one-step computation with $\rightarrow_{\mathcal{R}}$ represents a single step in a discrete-event simulation of a specification written in PMaude. For details about the admissibility assumptions on rewrite theories such as sort-decreasingness, operational termination, and confluence the reader is referred to [14]. Similarly, for details about one-step computation and sufficient conditions for absence of unquantified non-determinism in PMaude to [3]. The representation of BPMN processes in Section 4 satisfies these admissibility requirements and is free of unquantified non-determinism for valid BPMN processes.

Once a probabilistic system has been modeled in PMaude, various quantitative properties of the system can be specified by using the *Quantitative Temporal Expressions* language (QuaTEx) [3] and queried with the help of the PVeStA statistical model checker [5]. The QuaTEx language implements a framework for

parameterized recursive temporal operator definitions using a set of primitive non-temporal operators and the 'next' (◯) temporal operator. As an example, consider the following query over the process in Figure 2 that computes the probability that along a random path from a given state the insurance is bought within t days:

```
insuranceInTime(t)
   = if currentExecTime() > t then 0
     else if totalExecTime() > 0 then 0
          else if insuranceScheduled() > 0 then
                    if insuranceScheduled() < t then 1
                    else 0
               else ◯ insuranceInTime(t);
```

This defines the recursive operator `insuranceInTime(t)`, which returns 1 if along the execution path the insurance is bought no later than t time units and 0 otherwise. The (state) function `currentExecTime()` returns the global time associated with the state. The (state) function `totalExecTime()` returns the total final time of the simulation, or -1 if the simulation has not terminated. The (state) function `insuranceScheduled()` returns the time at which the next insurance buying task is scheduled; or -1 if none is scheduled. In this definition, the temporal operator ◯ takes an expression at the next state and makes it an expression for the current state. The following *state query* returns the expected number of times an insurance is bought within $t = 15$ days:

```
eval E[ ◯ insuranceInTime(15) ]
```

The expected value computed by this query is equal to the probability that along a random path, from the given initial state, the insurance is bought within 15 days. Note that this quantity lies in the $[0, 1]$ range because either the insurance is bought or not. As a preview of the type of analysis that can be performed on BPMN processes with the approach presented in this paper, this expression can be automatically quantified with PVeStA to be 0.81, with an error margin less than 0.01%. That is, insurance is bought in 81% of the cases.

This section ends with some insights on the use of Maude for the specification of PMaude systems (see, e.g., [3] for additional details). Maude's `random(n)` function provides a built-in purely functional pseudo-random number generator that returns the n-th random number following a uniform distribution. PMaude provides additional functions that build on this one to provide other random distributions. By default the seed 0 is used, but a different seed may be used by

providing appropriate command line options to Maude (see [14]). For running multiple executions, each of them is started with a different seed. Finally, instead of keeping a counter as part of our states, one can alternatively use the Maude predefined counter operator: "each time it has the opportunity to do a rule rewrite, it rewrites to the next natural number, starting at 0" [14]. This operator can indeed be used as implicit counter to generate consecutive random numbers. As explained in more detail in Section 4, the eval(SE) function is used to compute a random value relative to the stochastic expression SE. This function uses the counter operator to obtain the required probabilistic values, keeping it purely functional.

The reader is referred to [3] for additional details about QuaTEx syntax and semantics, how other logics such as the *Probabilistic Computation Tree Logic* (PCTL) [26] can be encoded in it, and the mechanisms used by PVeStA for statistical evaluation of QuaTEx expressions.

4. An Executable Specification in Rewriting Logic

This section presents an executable specification in rewriting logic for the fragment of BPMN enriched with timing behavior and probabilities described in Section 2. The focus in this section is twofold. On the one hand, it aims at helping the reader gain a high-level idea of the main features of the specification. On the other hand, it presents some rules and equations with some key features of the specification. Also mentioned in Section 1, additional details about the specification and examples can be found in the accompanying webpage [18].

The executable specification of BPMN is a probabilistic rewrite theory $\mathcal{R} = (\Sigma, E \uplus B, R)$. The equational subtheory $(\Sigma, E \uplus B)$ offers the infrastructure for defining a process in the sublanguage of BPMN described in Section 2, including the timing behavior for tasks and flows, and probabilities for outgoing flows of split gateways. The probabilistic rewrite rules R axiomatize how time advances and probabilistic choices are made in this infrastructure, in order for a given process to transition from an initial to a final state.

A system state in $\mathcal{R}$ can be seen as a tuple $(Proc, Toks, Sim)$, where $Proc$ is the set of nodes and sequence flows representing the graph structure of the process, $Toks$ is a *scheduler* implemented as a priority queue of tokens indicating which token is to be processed next, and Sim is a collection of variables and maps updated during the process execution with timing information. Overall, the pair $(Proc, Toks)$ defines the state of execution of a process $Proc$ with tokens $Toks$, and Sim includes auxiliary structures for timing simulation and stochastic analysis.

A one-step computation in $\mathcal{R}$ has the form

$$(Proc, Toks, Sim) \rightarrow_{\mathcal{R}} (Proc, Toks', Sim')$$

where the set of nodes and flows *Proc* remains unchanged, the set of tokens transitions to *Toks*′, and *Sim* transitions to *Sim*′.

Following the usual aproach in Real-Time Maude [41], all the actions in the system are instantaneous. Then time passing is modeled by a *tick* rule (see below) that is fired when no other actions can be performed, and results in reaching the next time at which further actions will become enabled. This is achieved by providing each token in the system with its time to completion. The *maximum time elapse* (mte) strategy advances time as much as possible so that no action is missed. Time passing is modelled by reducing the time to completion of every token in the same amount (using a *delta* function).

```
        < Proc : Process | >
        < PQ : Scheduler | tokens : Tks >
        < Sim : Simulation | gtime : T >
→       < Proc : Process | >
        < PQ : Scheduler | tokens : delta(Tks, T1)>
        < Sim : Simulation | gtime : (T + T1) >
if      T1 := mte(Tks) /\ T1 =/= 0
```

In Maude, it is not necessary to explicitly write all the objects' attributes. Only attributes that are used or modified need to appear in the left-hand side of the rule; attributes not appearing in the right-hand side are considered to be left unmodified.

4.1. State Representation

The term that represents the component *Proc* in the tuple (*Proc*, *Toks*, *Sim*) depends on the BPMN process under consideration and is automatically generated. The process transformation, which takes a BPMN process and generates the corresponding Maude code, is achieved by a plugin (a Python script) implemented as an extension of the VBPMN platform [31].

A process can abstractly be seen as a multiset of terms representing nodes and sequence flows, each of which has a unique identifier and has a time duration. In our proposal, a process is modeled as an object of class *Process*, with attributes

nodes and *flows* storing, respectively, the nodes and flows of a process. More precisely, there are five types of nodes (*start*, *end*, *task*, *split* and *merge*) and a single flow constructor (*flow*). The following set of terms represent, in the syntax of $\mathcal{R}$, the description of some of the elements of the process depicted in Figure 2 (please, notice the ellipses):

$\langle$ *Proc* : *Process* |
  nodes :
    (*start*(*initial*, *sf01*), *merge*(*g00*, *exclusive*, (*sf01*, *sf05*), *sf02*),
     task(*t0*, *"authorization"*, *sf02*, *sf03*, *Norm*(7, 1)),
     split(*g01*, *exclusive*, *sf03*, ((*sf05*, 0.4) (*sf06*, 0.1) (*sf1*, 0.5))), ...)
  flows :
    (*flow*(*sf01*, 0), *flow*(*sf02*, 0), *flow*(*sf03*, 0), *flow*(*sf05*, *Unif*(2, 3)), ...) $\rangle$

Figure 3 shows the process in Figure 2 labelled with the identifiers of tasks, gates and flows used in the above codification to facilitate its understanding. These terms represent the start event *initial* with outgoing flow *sf01*; the exclusive merge *g00* with incoming flows *sf01* and *sf05*, and outgoing flow *sf02*; the task *t0* with incoming flow *sf02*, outgoing flow *sf03*, and timing behavior *Norm*(7, 1); the exclusive split *g01* with incoming flow *sf03*, and outgoing flows *sf05* with probability 0.4, *sf06* with probability 0.1, and *sf1* with probability 0.5; and flows *sf01*, *sf02*, *sf03*, and *sf05* with duration 0, except for *sf05* whose duration is to be sampled uniformly from the interval $[2, 3]$.

In $\mathcal{R}$, there is support for the family of probability distribution functions listed in Section 2. Hence, time values can be computed using an *eval* function during execution according to these probability distribution functions.

The term *Toks* is a set of pairs of the form *token*(u, t) with u the identifier of the node/flow the token is at and t its run-to-completion time (a timer). Technically, it is a *scheduler* implemented as a class *Scheduler* with a list of tokens which is handled as a priority queue indicating which token is to be executed next depending on all run-to-completion times in the process. *Toks* is a crucial component in the executable specification $\mathcal{R}$ because it is used as the mechanism to avoid any form of unquantified non-determinism, so that the quantitative analysis performed on

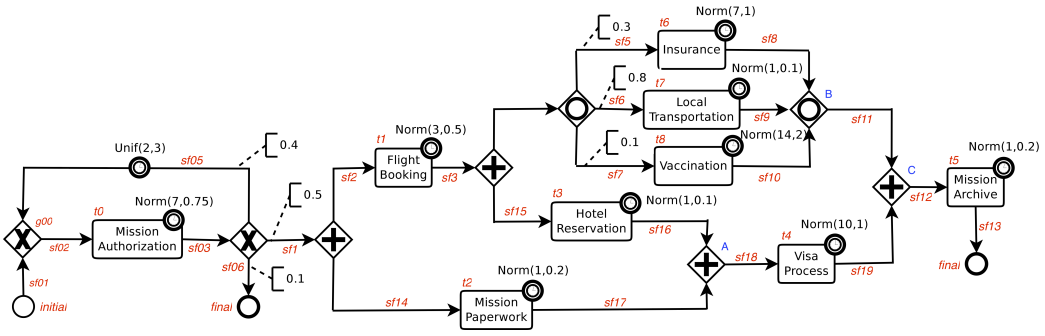

Figure 3: Decorated version of the process in Figure 2

$\mathcal{R}$ has a stochastic meaning (see Section 3). [1]

$$\langle\ PQ : Scheduler \mid tokens : (token(Id,\ T)\ \ldots)\ \rangle$$

The key idea is that there is no reachable state of the system in which two rules, or a same rule with two different matches, are applicable. The selection of the next action to be performed is totally determined by the scheduler. The action to be performed depends on the place of the token at the front of the scheduler. E.g., if the token at the front of the scheduler is at task *Id*, the rule handling tasks will be the only applicable rule for that task. It may happen that no action can be performed. E.g., a parallel merge requires tokens on all its incoming flows which may be not present yet. To avoid blocking situations in cases like this one, a *shifting* mechanism implemented on the scheduler is invoked. The shift operation is defined equationally, and, as we discuss below, the operation is invoked when a token cannot fire an action. When the token at the front of the scheduler is blocked, the shifting mechanism looks for the first non-blocking token in the queue and moves it to the front. This happens in parallel and inclusive merge gateways, where tokens for all or some of the incoming branches is expected. The combined application of the tick rule and the shifting mechanism eventually leads to the application of some action.

There are three key ingredients which ensure the specification to be free of unquantified non-determinism: (i) ensuring that the equational theory $(\Sigma, E \uplus B)$ is sort-decreasing, operationally terminating, and confluent (see Section 3); (ii) hav-

[1] Similar schedulers have previously been used to get specifications with no unquantified non-determinism in, e.g., [6, 21].

ing a fixed order for the outgoing flows of all split gateways, and (iii) guaranteeing that there is one single rule applicable at each time thanks to the scheduler that releases one token at a time.

An object of class *Simulation* is provided as the last component of the system state. It keeps information such as the global time of the simulation (*gtime*), time stamps (*tstamps*) for tasks and merge gateways, synchronization times (*syncs*) for each merge gateway, and sojourn times (*trips*) corresponding to the time it takes to each task to be executed. In other words, it stores the different values necessary for specific executions and for its real time analysis.

$$\langle\ Sim : Simulation \mid\ gtime : T,\ tstamps : \ldots,\ syncs : \ldots,\ trips : \ldots\ \rangle$$

4.2. State Transitions

The circulation of tokens along flows and tasks is modeled by the rules in $\mathcal{R}$. Each of the possible actions in a process is specified as a rewrite rule, with equations to specify auxiliary calculations. Some rules of $\mathcal{R}$ are introduced in this section to illustrate how state transitions are performed.

4.2.1. Beginning and End of Process Execution

A process is initiated when there is a token in its start node. When a process is setup in the provided Maude infrastructure, the first and only task in the scheduler is the start node of the process.

Given variables *NId* ranging over node identifiers, *FId* over flow identifiers, *Nodes* over sets of nodes, *Flows* over sets of flows, *Tks* over lists of tokens, *T* over time values, *TSs* over time stamps, i.e., maps from identifiers to time values, and *N* over natural numbers, the rule below specifies the execution of a start node. When a start node *NId* has a token with time zero at the front of the scheduler, the token is moved to its outgoing flow *FId* with the time resulting from the evaluation of the stochastic expression *SE* associated to that flow.

$$\begin{aligned}
&\langle\ Proc : Process \mid nodes : (start(NId,\ FId),\ Nodes),\\
&\qquad\qquad flows : (flow(FId,\ SE),\ Flows)\ \rangle\\
&\langle\ PQ : Scheduler \mid tokens : (token(NId,\ 0)\ Tks)\ \rangle\\
&\langle\ Sim : Simulation \mid gtime : T,\ tstamps : TSs\ \rangle\\
\rightarrow\ &\langle\ Proc : Process \mid\ \rangle\\
&\langle\ PQ : Scheduler \mid tokens : enqueue(Tks,\ token(FId,\ eval(SE)))\ \rangle\\
&\langle\ Sim : Simulation \mid tstamps : update(NId, T, TSs)\ \rangle
\end{aligned}$$

In the rule, a token is removed, another one is created, and the time stamps are updated. The time stamp of the process' initiation is the current global time T. The newly created token is assigned with its run-to-completion time by evaluating the stochastic expression *SE* associated to the chosen outgoing flow *FId*. The *eval* operator takes a stochastic expression and returns a time value, a rational number. If *SE* is a literal value, then this value is returned; otherwise, it is an expression involving a probabilistic function that is evaluated according to the given arguments. The *enqueue* operator inserts the new token in the priority queue in accordance with its time to completion. The *update* operator associates T to *NId* in the *TSs* map.

When a token reaches an end node, the execution time of the process is updated with the time elapsed from the process' initiation $TSs[init]$ and the current global time T. Such a behavior is specified by the following rule (operator $\ominus$ represents the 'monus' operation, i.e., subtraction over non-negative numbers):

$$
\begin{array}{rl}
 & \langle\ Proc : Process \mid nodes : (end(NId,\ FId),\ Nodes)\ \rangle \\
 & \langle\ PQ : Scheduler \mid tokens : (token(FId,\ 0)\ Tks)\ \rangle \\
 & \langle\ Sim : Simulation \mid gtime : T,\ tstamps : TSs,\ exec : T'\ \rangle \\
\rightarrow & \langle\ Proc : Process \mid\ \rangle \\
 & \langle\ PQ : Scheduler \mid tokens : Tks\ \rangle \\
 & \langle\ Sim : Simulation \mid exec : T \ominus TSs[init]\ \rangle
\end{array}
$$

Note that a process terminates when all tokens have reached an end event. This is detected during the simulation stage when the token queue is empty.

4.2.2. Tasks

The execution of tasks is modeled with two rewrite rules: when its incoming flow is completed, i.e., the flow has a token with time to completion 0, the token is moved to the task with a duration; when this timer reaches zero, the token is moved to the outgoing flow in a second rule.

The following rewrite rule specifies how tasks are initiated, where the variable *TName* ranges over task names, *FId1*, *FId2* over flow identifiers, and all other

variables are as above.

$$
\begin{array}{ll}
 & \langle\ \mathit{Proc} : \mathit{Process} \mid \mathit{nodes} : (\mathit{task}(\mathit{NId},\ \mathit{TName},\ \mathit{FId1},\ \mathit{FId2},\ \mathit{SE}),\ \mathit{Nodes})\ \rangle \\
 & \langle\ \mathit{PQ} : \mathit{Scheduler} \mid \mathit{tokens} : \mathit{token}(\mathit{FId1},\ 0)\ \mathit{Tks}\ \rangle \\
 & \langle\ \mathit{Sim} : \mathit{Simulation} \mid \mathit{gtime} : T,\ \mathit{tstamps} : \mathit{TSs}\ \rangle \\
\rightarrow & \langle\ \mathit{Proc} : \mathit{Process} \mid\ \rangle \\
 & \langle\ \mathit{PQ} : \mathit{Scheduler} \mid \mathit{tokens} : \mathit{enqueue}(\mathit{Tks},\ \mathit{token}(\mathit{NId},\ \mathit{eval}(\mathit{SE})))\ \rangle \\
 & \langle\ \mathit{Sim} : \mathit{Simulation} \mid \mathit{tstamps} : \mathit{update}(\mathit{NId},\ T,\ \mathit{TSs})\ \rangle
\end{array}
$$

This rule is enabled when the incoming flow *FId1* of task *NId* has been completed. The process state is updated by initiating the execution of the task *NId* with the time resulting from the evaluation of its stochastic expression *SE* (this is specified by adding $\mathit{token}(\mathit{NId},\ \mathit{eval}(\mathit{SE}))$ to the priority queue). The time stamp in the simulation is updated with the time at which the task *NId* is initiated, i.e., the current global time T.

The completion of a task is specified by the following rule, in which all variables are as above, and *SE1*, *SE2* range over stochastic expressions:

$$
\begin{array}{ll}
 & \langle\ \mathit{Proc} : \mathit{Process} \mid \mathit{nodes} : (\mathit{task}(\mathit{NId},\ \mathit{TName},\ \mathit{FId1},\ \mathit{FId2},\ \mathit{SE1}),\ \mathit{Nodes}), \\
 & \qquad\qquad \mathit{flows} : (\mathit{flow}(\mathit{FId2},\ \mathit{SE2}),\ \mathit{Flows})\ \rangle \\
 & \langle\ \mathit{PQ} : \mathit{Scheduler} \mid \mathit{tokens} : (\mathit{token}(\mathit{NId},\ 0)\ \mathit{Tks})\ \rangle \\
 & \langle\ \mathit{Sim} : \mathit{Simulation} \mid \mathit{gtime} : T,\ \mathit{tstamps} : \mathit{TSs},\ \mathit{trips} : \mathit{TSs'}\ \rangle \\
\rightarrow & \langle\ \mathit{Proc} : \mathit{Process} \mid\ \rangle \\
 & \langle\ \mathit{PQ} : \mathit{Scheduler} \mid \mathit{tokens} : \mathit{enqueue}(\mathit{Tks},\ \mathit{token}(\mathit{FId2},\ \mathit{eval}(\mathit{SE2})))\ \rangle \\
 & \langle\ \mathit{Sim} : \mathit{Simulation} \mid \mathit{trips} : \mathit{update}(\mathit{NId},\ T \ominus \mathit{TSs}[\mathit{NId}],\ \mathit{TSs'})\ \rangle
\end{array}
$$

Once enabled, this rule updates the state of the process by terminating the execution of the task *NId* and by initiating the execution of its outgoing flow *FId2*. The attribute *trips* in *Sim* keeps track of the execution time (i.e., sojourn time) of each task. This rule updates the execution time of the finishing task *NId* with the time elapsed from its initiation $\mathit{TSs}[\mathit{NId}]$ and the current global time T. The values T' and N' are computed in the same way as in the previous rule.

4.2.3. Gateways

The scheduler-based execution semantics of the subset of BPMN under consideration allows for unbalanced workflows, where there is not strict correspondence between splits and merges, as well as for looping behavior. In the rest of

this section, we present the encoding of the three kinds of gateways supported in this work (exclusive, parallel and inclusive), and we illustrate with a couple of rewrite rules (the reader interested in more details can consult the specification available online [18]).

The semantics of *exclusive gateways* is encoded with two rules, one rule for the split gateway and one rule for the merge gateway. The rule for the exclusive split gateway fires when a token with time zero is available in the input flow and generates a token for one of the output branches depending on the probabilities. The exclusive merge gateway executes when there is one token for one of the incoming flows. In that case, the token is consumed and a token is generated for the merge outgoing flow.

The following probabilistic rewrite rule specifies the behavior of an exclusive split gateway upon the arrival of a token. The variables *NId* range over node identifiers, *FId* over flow identifiers, *FIdPL* over lists of pairs (flow identifier, probability value), *Nodes* and *Flows* over sets of processes and flows, respectively, *Tk* over tokens, *Tks* over lists of tokens, and *Q* over floating-point numbers:

$$
\begin{aligned}
& \langle\ Proc : Process \mid nodes : (split(NId,\ exclusive,\ FId,\ FIdPL),\ Nodes), \\
& \qquad\qquad\qquad\qquad flows : Flows\ \rangle \\
& \langle\ PQ : Scheduler \mid tokens : (token(FId,\ 0)\ Tks)\ \rangle \\
\rightarrow\ & \langle\ Proc : Process \mid\ \rangle \\
& \langle\ PQ : Scheduler \mid tokens : enqueue(Tks,\ Tk)\ \rangle \\
\text{if}\ \ & Tk\ :=\ genTokExcSplit(FIdPL,\ Flows,\ unif(0,\ 1))
\end{aligned}
$$

This rule is enabled when the token at the incoming flow *FId* of the exclusive split *NId* is next in the scheduler *PQ* with run-to-completion time 0. The state of the process changes by updating the scheduler: such a token is consumed and the newly created token *Tk* is placed in the queue (note the use of the matching equation condition to assign to a fresh variable *Tk* the generated token). The choice of which outgoing flow in the exclusive split to assign the newly created token to is made by calling the auxiliary function *genTokExcSplit* with a uniformly sampled probability. This function is defined by the following two equations:

$$
\begin{aligned}
& genTokExcSplit(((FId,\ P)\ FIdPs),\ (flow(FId,\ SE),\ Flows),\ Q) \\
& \quad = \text{if } P > Q \text{ then } token(FId,\ eval(SE)) \\
& \qquad \text{else } genTokExcSplit(FIdPs,\ Flows,\ Q - P) \\
& genTokExcSplit(nil,\ Flows,\ Q) = nil
\end{aligned}
$$

Note that the first argument is a list and then the outgoing branches are always considered in the same order. The second argument is a set and therefore the stochastic expression of the chosen branch is selected by matching. By assuming that the probabilities associated to the outgoing flows of an exclusive split add up to 1, the function *genTokExcSplit* recursively selects the token with cumulative probability (in the order of the outgoing flows, which is always fixed) greater than Q.

The *genTokExcSplit* function iterates on each of the outgoing flows by evaluating its corresponding expression. It stops when the corresponding flow is found. As an example, consider the exclusive split *g01*, represented in Section 4.1 as

split(g01, exclusive, sf03, ((sf05, 0.4) (sf06, 0.1) (sf1, 0.5)))

Assuming that *unif(0, 1)* evaluates to 0.75, the following evaluation steps illustrate the behavior of function *genTokExcSplit* with the given arguments (note the ellipsis for the set of flows):

genTokExcSplit(((sf05, 0.4) (sf06, 0.1) (sf1, 0.5)), (...), 0.75)
= *genTokExcSplit(((sf06, 0.1) (sf1, 0.5)), (...), 0.35)*
= *genTokExcSplit(((sf1, 0.5)), (flow(sf01, SE), ...), 0.25)*
= *token(sf01, eval(SE))*

Parallel gateways transform quite easily in Maude. Indeed, the parallel split gateway rule is triggered when a token corresponding to the input flow is available. If so, the token is consumed and one token is added for each outgoing flow. The merge rule for the parallel gateway is executed when there is a token for each incoming branch. In that case, these tokens are removed and a new token is generated for the outgoing flow.

The following rule specifies the execution of a parallel merge gateway. The rule is fired when there is a token on some of the incoming branches of a parallel merge gateway with timer 0 at the front of the scheduler. Depending on whether tokens for all its branches are available or not (*allTokensParallel* operator), the gate is executed, removing the tokens in the incoming branches (*rmTks* operator) and adding a new token in the outgoing flow, or the shift operator is invoked. The shift function just traverses the queue of tokens until it finds the first one that may fire an action, moving it to the head of the queue, and thus enabling some other rule in the system.

$\langle$ *Proc* : *Process* | *nodes* : (*merge*(*NId*, *parallel*, (*FId1*, *FIds*), *FId*), *Nodes*),
flows : (*flow*(*FId*, *SE*), *Flows*) $\rangle$
$\langle$ *PQ* : *Scheduler* | *tokens* : (*token*(*FId1*, 0) *Tks*) $\rangle$
$\langle$ *Sim* : *Simulation* | *gtime* : *T*, *tstamps* : *TSs*, *syncs* : *TSs'* $\rangle$
$\rightarrow$ *if allTokensParallel*(*FIds*, *Tks*)
then $\langle$ *PId* : *Process* | $\rangle$
$\langle$ *PQ* : *Scheduler* |
tokens : *enqueue(rmTks(FIds, Tks), token(FId, eval(SE)))* $\rangle$
$\langle$ *Sim* : *Simulation* | *syncs* : *update*(*NId*, $T \ominus TSs[NId], TSs'$)$\rangle$
else shift($\langle$ *PId* : *Process* | $\rangle$
$\langle$ *PQ* : *Scheduler* | $\rangle$
$\langle$ *Sim* : *Simulation* | $\rangle$)

Inclusive gateways are the most complicated gateways to encode in Maude. An inclusive split gateway can trigger any number of outgoing flows (at least one). Similarly to the exclusive split gateway, we rely on a function that selects the set of outgoing flows to be triggered using the probability associated to each flow.

The semantics of inclusive merge gateways is quite intricate [13] and varies between versions 1.0 and 2.0 of BPMN. In $\mathcal{R}$, both versions of the inclusive merge gateways are supported; they are specified by several rules and equations. The following rule specifies the behavior of a merge gateway when used as a synchronization point (BPMN 1.0 semantics): it requires a backwards search analysis on the structure of the process and its current state of execution. In the following, the variables *NId* range over node identifiers, *FId1*, *FId2* over flow identifiers, *FIds* over lists of flow identifiers, *Nodes* and *Flows* over sets of processes and flows, respectively, *SE* over stochastic expressions, *Tks* over lists of tokens, T, T' over time values, *TSs*, *TSs'* over maps from identifiers to time values, and *N* over

natural numbers:

$$
\begin{aligned}
& \langle\ \mathit{Proc} : \mathit{Process} \mid \mathit{nodes} : (\mathit{merge}(\mathit{NId},\ \mathit{inclusive},\ (\mathit{FId1},\ \mathit{FIds}),\ \mathit{FId2}),\ \mathit{Nodes}), \\
& \qquad\qquad \mathit{flows} : (\mathit{flow}(\mathit{FId2}, \mathit{SE}),\ \mathit{Flows})\ \rangle \\
& \langle\ \mathit{PQ} : \mathit{Scheduler} \mid \mathit{tokens} : (\mathit{token}(\mathit{FId1},\ 0)\ \mathit{Tks})\ \rangle \\
& \langle\ \mathit{Sim} : \mathit{Simulation} \mid \mathit{gtime} : \mathit{T},\ \mathit{tstamps} : \mathit{TSs},\ \mathit{syncs} : \mathit{TSs'}\ \rangle \\
\rightarrow\ & \langle\ \mathit{Proc} : \mathit{Process} \mid \rangle \\
& \langle\ \mathit{PQ} : \mathit{Scheduler} \mid \mathit{tokens} : \mathit{enqueue}(\mathit{rmTks}(\mathit{Tks}, \mathit{FIds}),\ \mathit{token}(\mathit{FId2},\ \mathit{eval}(\mathit{SE})))\ \rangle \\
& \langle\ \mathit{Sim} : \mathit{Simulation} \mid \mathit{syncs} : \mathit{update}(\mathit{NId}, \mathit{T}, \mathit{TSs'})\ \rangle \\
\text{if}\ \ & \mathit{allTokensArrived}(\mathit{NId}, \mathit{Nodes}, \mathit{FIds}, \mathit{Tks})
\end{aligned}
$$

This rule is enabled when all previously created tokens at the inclusive split gateway have arrived to the inclusive merge gateway: the function *allTokensArrived* performs the above-mentioned backwards search analysis on the structure of the process and its current state of execution. This rule updates a process state in several ways. Although it is fired by the arrival of a token in one of its incoming flows, when all tokens have arrived, it removes from the priority queue the tokens associated to *all* incoming flows *FIds* of the inclusive merge gateway *NId*. This queue is also updated with the newly created token for the outgoing flow *FId2* with time the result of evaluating the stochastic expression *SE* associated to the outgoing flow *FId2*. The attribute *tstamps* keeps the time stamps of tasks and gateways that can later be used for stochastic analysis. The attribute *sync* keeps the time stamps of merge gateways: in this case, it is updated with the final synchronization time of the merge gateway *NId*. In the case in which not all necessary tokens have arrived, the shifting mechanism is invoked.

5. Stochastic Analysis of BPMN

In this section, the verification of processes using reachability analysis and model checking is presented. The focus then shifts to the stochastic analysis of timing properties using PMaude and PVeStA, with a specific application of those techniques to our running example. Finally, additional experiments on examples found in the literature are presented.

5.1. Simulation, Reachability, and Model Checking

Recall from Section 4 that a state in the rewriting logic specification $\mathcal{R}$ is a tuple of the form $(\mathit{Proc}, \mathit{Toks}, \mathit{Sim})$. While $(\mathit{Proc}, \_, \_)$ can be used to perform

static analysis on the structure of a BPMN process, (*Proc*, *Toks*, _) can be used to perform reachability analysis and more general LTL model checking [15, 22].

Simulation is very useful for exploring system executions. In Maude, simulation relies on rewriting, which consists in successively applying equations and rewrite rules on an initial term (a BPMN process here), with the possibility of using some strategy language to guide the execution. Since systems may be rewritten in many different ways, Maude also provides a *search* command, which allows us to explore the reachable state space up to a certain depth. Thus, we can perform analysis on the reachability of states satisfying certain conditions, *e.g.*, when searching for deadlock states or other undesired situations. For example, given our running example in Figure 2, and its corresponding Maude representation InitSystem introduced in Section 4.1, the following search command checks that there is no reachable final state with tokens in it, which shows that there is no deadlock.

```
> search InitSystem =>! Conf such that getNumToks(Conf) =/= 0 .
No solution.
```

Notice the use of '=>!' to limit the check to final states. The search command allows us to search for states with any given pattern satisfying any given conditions on it. In this case we use the auxiliary function getNumToks(Conf) to search for states with zero tokens.

Other analysis tools available in the Maude system can also be used. For instance, Maude's Linear Temporal Logic (LTL) explicit-state model checker [22] can be used for analyzing LTL formulas on a business process. Maude's model checker allows one to check whether every possible behavior starting from a given initial state (the start node in BPMN) satisfies a given LTL property. In general, any safety and liveness property of a system can be decided with this model checker when the set of states reachable from an initial state is finite. Full verification of invariants in infinite-state systems can be accomplished by verifying them on finite-state abstractions [38] of the original infinite-state system, that is, on an appropriate quotient of the original system whose set of reachable states is finite. In the context of this paper, beyond classic properties such as deadlock-freeness, the properties that can be verified depend on the example and should be specified by the developer, e.g., a certain task is always achieved after another specific task. In order to make the property writing easier, the developer can rely on well-known patterns as those presented in [25, 30] for timing properties.

For instance, given propositions FlightBooking and VisaProcess, which are true in states in which the process is executing these respective tasks, *i.e.*, there is a token in the corresponding task, we can check that the visa request is always

processed after a flight booking as follows:

```
> reduce modelCheck(InitSystem,
            [](FlightBooking -> <> VisaProcess)) .
result Bool: true
```

5.2. Timing Properties

This section explains how predictive performance evaluation of BPMN processes can be achieved with stochastic analysis of timing properties by using PMaude and PVeStA. To that goal, the *Simulation* object plays a key role. This object has a multiset of variables and mappings with two main purposes: on the one hand, it maintains the timing attributes of a process that are collected during simulation; on the other hand, it records the values needed for the discrete-event simulation of the process.

This subsection focuses on measuring two timing properties of a process, namely, the *expected processing time* (or latency) and the *expected synchronization time* for each one of the merge gateways. The expected processing time of a process is the time it takes all the tokens to reach an end event from the initial event. This measure is calculated for the entire process. The expected synchronization time is the time between the arrival of the first token through one of the incoming flows of a merge gateway and the activation of the gateway. The synchronization time for exclusive merge gateways is always zero. This time is also zero for inclusive gateways when only one of the flows is active. This measure is calculated for each one of the merge gateways of a process. These quantities are computed using the statistical model checker PVeStA, which performs Monte Carlo simulations and returns the expected value for each measure. That is, PVeStA calculates the average of the values of each measure for all the executions. It also provides the variance and error of the results. Since all the values are available, minimum and maximum values (relative to the simulations) are also computed, providing a range of values for the expected values.

PVeStA offers two interesting ways of minimizing the time required for the analysis when needed: (i) the possibility of specifying different parameters for the analysis such as error bounds, and (ii) support for parallel processing of the Monte Carlo simulations. Depending on the error bounds specified, PVeStA will need more or less samples. Technically, (i) results in a trade-off between computational time and precision, while (ii) results in a trade-off between computational time and availability of nodes for performing the simulations. Tables 1 and 2 show results on experiments using different error bounds and number of servers. Table 1

Error bound	Expected proc. time (days)	Samples generated	Analysis time (seconds)
0.00001	25.43	19500	59.6
0.0001	25.67	14100	43.8
0.001	25.32	9000	28.0
0.01	25.63	3900	12.1

Table 1: Results for different error bounds

Number of servers	Expected proc. time (days)	Samples generated	Analysis time (seconds)
1	25.60	8700	67.0
2	25.41	8400	38.4
3	25.31	9000	31.2
4	25.57	9000	28.5

Table 2: Results for different numbers of servers

shows the expected processing times, number of required samples, and analysis time of the case study for different error bounds. By increasing the error bound, the expected time remains very similar. However, it takes much less samples to obtain the result and the time for computing these results significantly decreases. Notice that the expected processing times are non-monotonic, which is sound with the algorithms implemented in PVeStA; please see [46] for details on how the errors bounds and other parameters are used in the tool.

By using several nodes, the performance gains are also high. Table 2 shows the results of the analysis using 1, 2, 3, and 4 servers working in parallel. The tool does not make any assumption on the underlying architecture. In this case the expected processing times are non-monotonic either. The same error was used in all the experiments, all lying in the range of acceptable values.

Quantitative Analysis of the Running Example. Table 3 shows the minimum, maximum, variance, and expected processing time for the running example (Figure 2). Table 4 shows the minimum, maximum, variance, and expected synchronization times for each one of the merge gateways in the running example. In this process, gateways are identified as A, B, and C in Figure 2. The analysis time is just 7.8 seconds for Gate A. However, it takes more than 150 seconds for Gates B and C. The greater variability in the results requires a much bigger number of samples (46500 vs. 2100). By looking at the expected synchronization times of merge gateways (Table 4), some interesting observations can be made. Note that the maximum times recorded during the Monte Carlo simulations are much

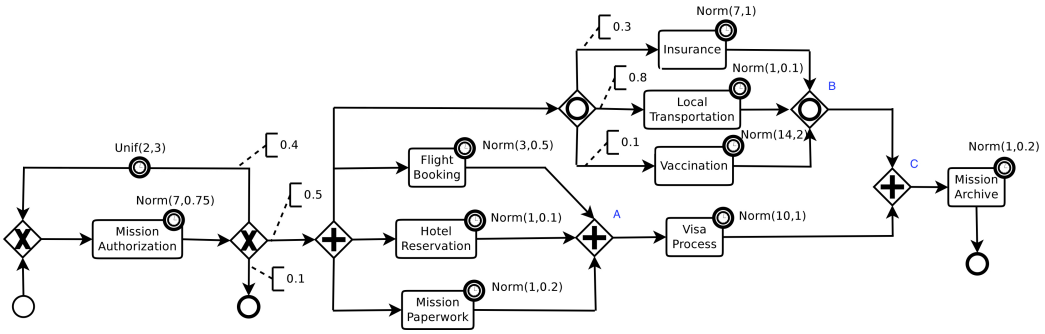

Figure 4: Re-designed version of the running example.

higher than the expected times, particularly for B and C, because of the duration of the vaccination task ($Norm(14, 2)$). This observation can also explain why the expected synchronization time for B is high. Furthermore, the difference between expected and maximum synchronization times for A and C is also high. From this information, this process can be improved by a simple re-design. For instance, the hotel reservation and the flight booking can be done in parallel. The resulting process is depicted in Figure 4.

Size					Processing time (s)				Samples	Analysis
Tasks	Flows	✕	+	○	Min	Max	Var	Exp	generated	time (s)
9	24	2	4	2	4.81	104.07	81.69	25.39	8400	25.5

Table 3: Experimental results for the running example

Merge gateway	Synchronization times (days)				Samples generated	Analysis time (s)
	Min	Max	Var	Exp		
A	1,13	4,70	0.28	2.99	2100	7.8
B	0	19,89	23.15	4.96	46500	165.6
C	0,01	13,31	4.88	2.36	46500	156.2

Table 4: Synchronization times for the merge gateways in the running example

Table 5 compares the stochastic results for the two versions of the running example. These results show that, for instance, the expected processing time decreases from 25.39 to 24.18 days in the re-designed process. This improvement is confirmed by Table 6 because, despite the fact that the maximum and average synchronization times for gateway C in the second version of the process increase, all other times decrease. This means that completing the flight booking before

Running example	Size					Processing time			
	Tasks	Flows				Min	Max	Var	Exp
V1 (Fig. 2)	9	24	2	4	2	4.81	104.07	81.69	25.39
V2 (Fig. 4)	9	24	2	3	2	4.81	73.21	75.67	24.18

Table 5: Comparative sizes and processing times for the running examples

Merge gateway	Synchronization time V1				Synchronization time V2			
	Min	Max	Var	Exp	Min	Max	Var	Exp
A	1,13	4,70	0.28	2.99	0.34	4.01	0.29	2.09
B	0	19,89	23.15	4.96	0	19.69	22.95	4.95
C	0,01	13,31	4.88	2.36	0.01	15.31	5.68	3.56

Table 6: Comparative synchronization times for the running examples

starting the hotel reservation, insurance, local transportation, and vaccination was penalizing the expected processing time. In other words, since the second version of the process reduces the number of sequential dependencies by allowing more parallelism, a slight decrease in expected processing time and synchronization times can be observed.

5.3. Other Experiments

Table 7 summarizes experimental results on the stochastic analysis of timing properties of more than 30 BPMN processes. Our original motivation was to only use in our experiments real-world processes found in the literature. However, most of them were quite small in size and exhibiting very simple patterns. Therefore, we complemented these processes with other processes we designed ourselves where we tried to cover the BPMN expressiveness supported in this paper (different gateways possibly nested, loops, unbalanced structure, etc.) and where some of these processes involved a significant number of nodes (40 tasks and about 25 gateways for processes p0047-i and p0047-g). All these processes are available online at [18]. Specifically, this table provides measures of the complexity (in size), estimated processing time, and the analysis effort. Complexity is measured by the number of tasks, gateways, flows, and loops in the processes. The processing time calculated for each process are the minimum and maximum times recorded during the simulations, and the expected processing time of the process. For the analysis effort, the number of Monte Carlo runs and their total computation time is provided.

First of all, the analysis time presented in this table is always reasonable going from a few seconds to a few minutes. Since those results are computed us-

BPMN Processor	Complexity						Processing time			Analysis effort	
	Tasks	Flows	✖	✚	○	Loops	Minimum	Maximum	Expected	Samples	Time (sec)
Account opening [43]	15	29	3	2	2	0	70	150	94	1200	11.23
Account opening	15	28	2	2	2	0	93	712	354	37800	132.70
Account opening V2	15	29	3	2	2	0	563	766	651	900	3,79
Account opening V3	16	32	4	2	2	1	539	1643	549	1200	4.91
Account opening V4	16	32	5	2	2	1	539	1430	765	7200	31.64
Account opening V5	16	33	5	2	2	1	481	1796	774	4800	21.49
Apartment lease [47]	8	17	0	5	0	0	3963	3963	3963	300	2.64
Booking sys. [42]	6	11	2	0	0	1	30	200	50	12000	35.39
Car assembling [44]	11	18	2	2	0	0	80	80	80	300	1.91
Citizen migration [10]	4	11	2	2	0	0	6317	6317	6317	300	1.66
f0001	3	10	4	0	0	0	4	6	4	2100	5.01
Incident management	7	16	5	0	0	0	398	585	453	1500	3.40
Incident management V2	8	22	8	0	0	0	394	745	506	3600	10.74
Inclusive retry	4	12	2	0	2	1	120	1946	375	42300	116.81
Leave management	4	7	1	0	0	0	259	290	275	300	0.75
Leave management V2	6	13	3	0	0	1	205	1175	293	11700	30.53
Lunch payment	6	24	8	0	0	3	194	1219	304	11400	29.72
Parallel retry	4	12	2	2	0	0	158	3056	363	42600	160.80
Publishing system	12	30	6	2	2	2	311	1676	550	9600	40.00
Publishing system V2	12	32	7	2	2	2	342	1200	562	5400	30.79
Publishing system V3	12	33	7	2	2	2	305	1335	551	5700	28.45
Publish whitepaper	6	16	2	2	0	2	0	259	64	46500	124.51
p0047-i	40	87	12	9	4	0	7230	9160	7885	300	5.44
p0047-g	40	87	10	9	6	0	7230	9160	7959	300	5.72
p0058-a	16	31	0	0	2	0	3006	3661	3457	300	2.33
Release baseline [23]	9	25	8	2	0	2	10	550	81	42000	194.82
Retry system	2	6	2	0	0	1	85	2260	230	46500	79.21
Retry system V2	2	8	3	0	0	1	141	907	210	13800	25.48
Shipment [34]	8	18	2	2	2	0	40	40	40	300	1.88
Shipment [9]	8	18	2	2	2	0	353	439	365	600	3.43
Software development [40]	6	19	7	0	0	1	175	2263	484	12300	41.84

Table 7: Other experimental results with processes found in the literature.

ing Monte Carlo simulations (and not exhaustive explorations), the number of tasks/flows and the structure of the process (number of parallel and inclusive gateways, presence of loops) does not impact the analysis time, see, e.g., examples p0047-i and p0047-g. The main factor of increase of the analysis time comes from the number of samples required to obtain the results, see for instance the example 'parallel retry', which deserves more than 40000 samples for obtaining the processing time, then resulting in an analysis time of more than 2 minutes. The reason for this is the way PVeStA works. It keeps running simulations until the specified error bound is satisfied. In those cases in which the data shows a high variance, more samples are needed to meet the error bound. The example 'apartment lease' exhibits the same minimum, maximum, and average times because it involves only parallel gateways, and in that case, all behaviors are systematically

executed.

Some other observations can be made on this data. Taking a look at the different variants of the account opening example, one can see how the number of gates and loops, i.e., the complexity of the process, is the main factor determining the number of samples required and then justifying the analysis time. The probabilities assigned to each of the branches in a split gateway are also relevant. A loop that happens with a low probability may have an insignificant impact on the computation. See, e.g., the "Lunch payment" case, where, in spite of its eight exclusive gates and three loops, the computation time is quite low. One last observation is that most of the examples found in the literature have a low complexity, and the most complicated cases have been artificially introduced by the authors in order to evaluate the proposal.

6. Related Work

Several works propose extensions of BPMN with time constructs, see, e.g. [7, 24]. In [24], the authors present Time-BPMN, an extension of BPMN to represent various constraints and dependencies that may arise when modelling business processes. The temporal constraints are used to control the beginning/end of an activity, whereas temporal dependencies involve two activities and indicate the relationship between their respective beginning/end. In [7], a metamodel-based approach to integrate temporal constraints and dependencies is introduced. The time aspects are specified using rules and OCL constraints capture the semantics of these rules. [24] introduces time as a first class citizen to the BPMN standard whereas [7] proposes an MDE-based approach that enrich the existing BPMN control-flow graph model. In comparison, the goal in the present paper was not to extend the BPMN notation to take expressive modeling of time constraints into account, but to rely on usual time representation (duration of task and flow) and concentrate on the analysis of key timing properties in these models.

Several authors have used rewriting logic and Maude to model and analyze BPMN processes. El-Saber and Boronat [23] propose a translation of BPMN into rewriting logic with a special focus on data-based decision gateways. They provide mechanisms to avoid structural issues in workflows such as flow divergence by introducing the notion of "well-formed" BPMN process. Their approach aims at avoiding incorrect patterns by syntactic analysis. [23] focuses on behavioral constructs but does not support time features. Kheldoun *et al.* [29] propose high-level Petri nets and to use Maude's LTL model checker for, respectively, specifying BPMN processes and analyzing behavioral properties. They also focus on

handling exceptions and activity cancellation, but do not support any notion of time. In our approach, compared to [23, 29], we chose to support probabilities instead of data-based conditions. Corradini et al. present in [16] BProVe, a tool for the verification of business processes modeled in BPMN. The tool accepts BPMN processes in standard notation and can perform checks of soundness and safeness on them, as defined in [52] and [17], respectively, using Maude's LTL model checker.

There is a significant effort aimed at providing formal semantics and verification techniques for business processes using Petri nets (see, e.g., [10, 17, 44]). [10] presents a business process monitoring platform that relies on the ProM mining tool and Petri nets tools for verifying synchronization, sojourn and waiting times. Raedts *et al.* [44] translate BPMN processes into Petri nets and use model checkers to analyze invariants such as deadlock freedom. Compared to that line of research, the executable specification in rewriting logic presented in Section 4 can be seen as a semantic framework for BPMN, yet it is not the primary goal of this paper. The main difference with respect to those works is that the focus here is in the formal specification and verification of quantitative aspects of processes.

Another research direction focuses on the use of process algebras for formalizing and verifying BPMN processes. The authors of [49] present a formal semantics for BPMN by an encoding into the process algebra CSP. They show in [50] how such a semantics can be used to verify compatibility between business participants in a collaboration. There is a further extension in [51] to propose a timed semantics of BPMN with delays. In [12, 35, 36], the focus is on the semantics formalized in [49, 51] by proposing an automated transformation from BPMN to timed CSP, as well as composition verification techniques for checking properties using the FDR2 model checker. Poizat *et al.* [43] present an encoding of an untimed subset of BPMN into the LNT process algebra for supporting the analysis of process evolution.

Herbert and Sharp [27] propose an algorithm for translating a BPMN subset extended with probabilistic information into the guarded command language used by PRISM. This enables model checking of quantitative properties of business processes such as transient probabilities, occurrence of events, and best-/worst-case scenarios. Oliveira *et al.* [40] present an approach based on Generalized Stochastic Petri Nets (GSPN) for performance evaluation of business workflows. That work has a particular focus on BPEL and provides a tool to transform BPEL models to GSPN. The analysis techniques focus on processing times and costs, which are computed by simulation (systems are assumed to be bounded) using the TimeNET toolkit. Braghetto *et al.* [9] propose an approach for analytical per-

formance evaluation of business processes using a conversion of BPMN diagrams to Stochastic Automata Network (SAN) models. With their approach, processing times can be analyzed, as well as how resources can be impacted by workload. The work presented in this paper differs by the expressiveness of the supported subset of BPMN (e.g., unbalanced workflows are not supported by [9] and inclusive merge are not supported by [27]) and by the kind of properties that can be analyzed.

The expected processing time, the expected synchronization time for merge gateways, and other similar measures have been considered by some authors to evaluate deployed processes by, e.g., collecting timestamps from logs (see, e.g., [1, 2, 10]). Task duration has been specified using literal values in [9, 10, 40] and probabilities have been used to specify the likelihood of branching in [9, 40]. Inclusive gateways are only supported in [9, 34, 43] and loops only in [23, 40, 42]. The goal in most of these works is, however, quite different. Overall, Oliveira *et al.* [40] possibly have the closest approach to the one presented in this paper: task durations and probabilities for exclusive branching can be specified, and loops and continuous time are considered. However, they cannot specify durations with stochastic expressions.

The model checking problem for the stochastic extension of BPMN presented in this paper has been approached by statistical model checking (SMC) (see [4] for a recent survey). In particular, the BPMN extension has been specified as a purely probabilistic rewrite system, in the PMaude language [3], that can be analyzed by finitely many runs with the help of PVeStA [5]. In this approach, hypothesis testing is used to infer whether these runs provide statistical evidence for the satisfaction or violation of *quantitative* properties in terms of probabilistic guarantees (i.e., subject to some error threshold). Other frameworks for SMC such as Prism [32] were considered for supporting the statistical analysis presented in this work. However, due to the availability of PMaude and its tight integration with PVeStA, and also because of the previous experience with algebraic specification, the authors were inclined to use rewriting logic. In contrast to numerical model checking algorithms that incrementally compute the exact/approximate measure of paths satisfying a given property (see, e.g., [8]), the SMC approach enjoys the advantage of avoiding the state explosion problem and thus can be applied to larger classes of systems. A drawback of the SMC approach is that the required sample size of runs grows quadratically with respect to the required confidence of the result [33]. However, the experiments presented in this work have all executed within reasonable time limits. The reader is referred to [4, 33] for a comprehensive survey of approaches and tools for SMC.

This paper is an extended version of a conference paper published in [20]. The key additions of this journal version are as follows: (i) the BPMN notation is extended with probability distribution functions for specifying the duration of tasks and flows, and with probabilities for describing the behavior of split gateways; (ii) the Maude rewrite theory was enhanced to take these new elements into account; (iii) due to the presence of probabilities, the PVeStA statistical model checker was used instead of classic Maude analysis tools for computation of the verification results; (iv) the approach was used on a large number of case studies; and (v) the review of related work was updated and increased.

7. Concluding Remarks

This paper presents how stochastic analysis of BPMN timing properties can be carried out using rewriting logic. In this approach, stochastic expressions for specifying time (duration) and branching behavior are associated to a process. The stochastic analysis offers support for processes with: tasks and flows whose times can be constant values or sampled from several probabilistic distribution functions; probabilistic inclusive and exclusive gateways as well as parallel gateways; uniformly; looping behaviors; and unbalanced workflow structures. The specification of BPMN is an executable rewrite theory in the Maude rewrite system and statistical model checking is automatically performed with PVeStA. The approach presented in this paper has been applied to more than 30 examples, with a high number of them borrowed from the literature, by analyzing expected processing times of processes and expected synchronization times at merge gateways. This approach can be useful for refining designs with the ultimate goal of reducing operational costs.

Future work is in two directions. One perspective is to allow BPMN specifications with resources and multiple instances of a process. This would enable a more general type of stochastic verification for quantitative analysis of BPMN processes. A second perspective is to extend the fragment of BPMN used in this paper with data, supported by the newly developed rewriting modulo SMT technique [45]. A first step towards this goal has been presented at [19], where rewriting modulo SMT is used for the analysis of BPMN processes with data and interacting with the environment. One interesting technical challenge is related to the shifting mechanism. Although it has been extensively validated using reachability analysis and model checking, it is the authors' aim to prove that the shifting mechanism does not introduce deadlocks or livelocks for any non-blocking process.

Acknowledgments. The authors would like to thank the anonymous reviewers for their valuable comments on an earlier draft of this paper. F. Durn has been partially supported by Spanish MINECO/FEDER project TIN2014-52034-R. The work of C. Rocha was partially supported by CAPES, Colciencias, and INRIA via the STIC AmSud project "EPIC: EPistemic Interactive Concurrency" (Proc. No 88881.117603/2016-01), and by Capital Semilla 2017, project "SCORES: Stochastic Concurrency in Rewrite-based Probabilistic Models" (Proj. No. 020100610).

References

[1] van der Aalst, W.M.P., 2016. Process Mining - Data Science in Action, 2nd Ed. Springer.

[2] van der Aalst, W.M.P., van Dongen, B.F., 2002. Discovering Workflow Performance Models from Timed Logs, in: Han, Y., Tai, S., Wikarski, D. (Eds.), Proc. of EDCIS, Springer. pp. 45–63.

[3] Agha, G., Meseguer, J., Sen, K., 2006. PMaude: Rewrite-based Specification Language for Probabilistic Object Systems. ENTCS 153, 213 – 239.

[4] Agha, G., Palmskog, K., 2018. A survey of statistical model checking. ACM Transactions on Modeling and Computer Simulation (TOMACS) 28, 6:1–6:39. doi:10.1145/3158668.

[5] AlTurki, M., Meseguer, J., 2011. PVeStA: A Parallel Statistical Model Checking and Quantitative Analysis Tool, in: Proc. of CALCO, Springer. pp. 386–392.

[6] AlTurki, M., Meseguer, J., Gunter, C.A., 2009. Probabilistic modeling and analysis of dos protection for the ASV protocol. Electr. Notes Theor. Comput. Sci. 234, 3–18.

[7] Arevalo, C., Cuaresma, M.J.E., Ramos, I.M., Domínguez-Muñoz, M., 2016. A metamodel to integrate business processes time perspective in BPMN 2.0. Information & Software Technology 77, 17–33.

[8] Baier, C., Haverkort, B., Hermanns, H., Katoen, J.P., 2003. Model-checking algorithms for continuous-time markov chains. IEEE Transactions on Software Engineering 29, 524–541. doi:10.1109/TSE.2003.1205180.

[9] Braghetto, K.R., Ferreira, J.E., Vincent, J.M., 2011. Performance Evaluation of Business Processes through a Formal Transformation to SAN, in: Proc. of EPEW, Springer. pp. 42–56.

[10] Bruni, R., Corradini, A., Ferrari, G.L., Flagella, T., Guanciale, R., Spagnolo, G., 2011. Applying Process Analysis to the Italian eGovernment Enterprise Architecture, in: Proc. of WS-FM, Springer. pp. 111–127.

[11] Bruni, R., Meseguer, J., 2006. Semantic Foundations for Generalized Rewrite Theories. Theoretical Computer Science 360, 386–414.

[12] Capel, M., Mendoza-Morales, L., 2012. Automating the Transformation from BPMN Models to CSP+T Specifications, in: Proc. of SEW, IEEE. pp. 100–109.

[13] Christiansen, D.R., Carbone, M., Hildebrandt, T.T., 2011. Formal Semantics and Implementation of BPMN 2.0 Inclusive Gateways, in: Proc. of WS-FM, Springer. pp. 146–160.

[14] Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C., 2007a. All About Maude - A High-Performance Logical Framework. volume 4350 of *LNCS*. Springer.

[15] Clavel, M., Durán, F., Hendrix, J., Lucas, S., Meseguer, J., Ölveczky, P.C., 2007b. The Maude Formal Tool Environment, in: Proc. of CALCO, Springer. pp. 173–178.

[16] Corradini, F., Fornari, F., Polini, A., Re, B., Tiezzi, F., Vandin, A., 2017. BProVe: A Formal Verification Framework for Business Process Models, in: Proc. of ASE, IEEE Computer Society. pp. 217–228.

[17] Dijkman, R., Dumas, M., Ouyang, C., 2008. Semantics and Analysis of Business Process Models in BPMN. Inf. Softw. Technol. 50, 1281–1294.

[18] Durán, F., Rocha, C., Salaün, G., 2017. A Note on Stochastic Analysis of BPMN Timing Properties in Rewriting Logic. URL: `http://maude.lcc.uma.es/BPMN-P`.

[19] Durán, F., Rocha, C., Salaün, G., 2018. Symbolic specification and verification of data-aware BPMN processes using rewriting modulo SMT, in: Proc. of WRLA.

[20] Durán, F., Salaün, G., 2017. Verifying Timed BPMN Processes using Maude, in: Proc. of COORDINATION, Springer. pp. 219–236.

[21] Eckhardt, J., Mühlbauer, T., AlTurki, M., Meseguer, J., Wirsing, M., 2012. Stable availability under denial of service attacks through formal patterns, in: de Lara, J., Zisman, A. (Eds.), Proc. of FASE, Springer. pp. 78–93.

[22] Eker, S., Meseguer, J., Sridharanarayanan, A., 2002. The Maude LTL Model Checker, in: Proc. of WRLA, Elsevier. pp. 115–142.

[23] El-Saber, N., Boronat, A., 2014. BPMN Formalization and Verification using Maude, in: Proc. of BM-FA, ACM. pp. 1–8.

[24] Gagné, D., Trudel, A., 2009. Time-BPMN, in: Proc. of CEC'09, IEEE Computer Society. pp. 361–367.

[25] Gruhn, V., Laue, R., 2006. Patterns for Timed Property Specifications. ENTCS 153, 117–133.

[26] Hansson, H., Jonsson, B., 1994. A Logic for Reasoning about Time and Reliability. Formal Aspects of Computing 6, 512–535.

[27] Herbert, L., Sharp, R., 2012. Using Stochastic Model Checking to Provision Complex Business Services, in: Proc. of HASE, IEEE. pp. 98–105.

[28] ISO/IEC, 2013. International Standard 19510 – Business Process Model and Notation.

[29] Kheldoun, A., Barkaoui, K., Ioualalen, M., 2015. Specification and Verification of Complex Business Processes - A High-Level Petri Net-Based Approach, in: Proc. of BPMN, Springer. pp. 55–71.

[30] Konrad, S., Cheng, B.H.C., 2005. Real-time Specification Patterns, in: Proc. of ICSE'05, ACM. pp. 372–381.

[31] Krishna, A., Poizat, P., Salaün, G., 2017. VBPMN: Automated verification of BPMN processes, in: Proc. of IFM'17, Springer.

[32] Kwiatkowska, M., Norman, G., Parker, D., 2011. PRISM 4.0: Verification of probabilistic real-time systems, in: Gopalakrishnan, G., Qadeer, S. (Eds.), Proc. 23rd International Conference on Computer Aided Verification (CAV'11), Springer. pp. 585–591.

[33] Larsen, K.G., Legay, A., 2014. Statistical model checking past, present, and future, in: Margaria, T., Steffen, B. (Eds.), Proc. of ISoLA(2)'14, Springer. pp. 135–142.

[34] Mateescu, R., Salaün, G., Ye, L., 2014. Quantifying the Parallelism in BPMN Processes using Model Checking, in: Proc. of CBSE, ACM. pp. 159–168.

[35] Mendoza-Morales, L., Capel, M., Pérez, M., 2010. A Formalization Proposal of Timed BPMN for Compositional Verification of Business Processes, in: Proc. of ICEIS, Springer. pp. 388–403.

[36] Mendoza-Morales, L., Capel, M., Pérez, M., 2012. Conceptual Framework for Business Processes Compositional Verification. Inf. & Sw. Techn. 54, 149–161.

[37] Meseguer, J., 1992. Conditional Rewriting Logic as a Unified Model of Concurrency. Theoretical Computer Science 96, 73–155.

[38] Meseguer, J., Palomino, M., Martí-Oliet, N., 2003. Equational Abstractions, in: Proc. of CADE'03, Springer. pp. 2–16.

[39] Object Management Group, 2011. Business Process Model and Notation – V. 2.0.

[40] Oliveira, C., Lima, R., Andre, T., Reijers, H., 2009. Modeling and Analyzing Resource-constrained Business Processes, in: Proc. of SMC, IEEE. pp. 2824–2830.

[41] Ölveczky, P.C., Meseguer, J., 2007. Semantics and Pragmatics of Real-Time Maude. Higher-Order and Symbolic Computation 20, 161–196.

[42] Poizat, P., Salaün, G., 2012. Checking the Realizability of BPMN 2.0 Choreographies, in: Proc. of SAC, ACM. pp. 1927–1934.

[43] Poizat, P., Salaün, G., Krishna, A., 2016. Checking Business Process Evolution, in: Proc. of FACS, Springer. pp. 36–53.

[44] Raedts, I., Petkovic, M., Usenko, Y.S., van der Werf, J.M., Groote, J.F., Somers, L., 2007. Transformation of BPMN Models for Behaviour Analysis, in: Proc. of MSVVEIS, pp. 126–137.

[45] Rocha, C., Meseguer, J., Muñoz, C., 2017. Rewriting Modulo SMT and Open System Analysis. J. of Logical and Algebraic Methods in Programming 86, 269 – 297.

[46] Sen, K., Viswanathan, M., Agha, G., 2005. On statistical model checking of stochastic systems, in: Etessami, K., Rajamani, S.K. (Eds.), Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings, Springer. pp. 266–280.

[47] Sun, Y., Su, J., 2011. Computing Degree of Parallelism for BPMN Processes, in: Proc. of ICSOC, Springer. pp. 1–15.

[48] Walck, C., 2007. Handbook on Statistical Distributions for Experimentalists. Technical Report SUF–PFY/96–01. Universitet Stockholms. Stockholm.

[49] Wong, P., Gibbons, J., 2008a. A Process Semantics for BPMN, in: Proc. of ICFEM, Springer. pp. 355–374.

[50] Wong, P., Gibbons, J., 2008b. Verifying Business Process Compatibility, in: Proc. of QSIC, IEEE. pp. 126–131.

[51] Wong, P.Y.H., Gibbons, J., 2009. A Relative Timed Semantics for BPMN. ENTCS 229, 59–75.

[52] Wynn, M.T., Verbeek, H.M.W., van der Aalst, W.M.P., ter Hofstede, A.H.M., Edmond, D., 2009. Business Process Verification - Finally a Reality! Business Process Management Journal 15, 74–92.